INF8215: Travail pratique # 1
RushHour

Travail présenté par
Rafael Blanchet-Lecomte, # 1678307
Vincent Audet, # 1637998

Le dimanche 19 février 2017
École Polytechnique Montréal

Représentation du problème

La première partie du travail consistait simplement en l'implémentation de deux méthodes. Tout d'abord, il nous était demandé de compléter la méthode *success* de la classe *State* qui déterminait si l'état dans lequel on se trouvait était final. Pour ce faire, et puisque la voiture rouge ne pouvait se retrouver que sur la troisième ligne (ligne avec la sortie), nous n'avions qu'à vérifier si la position de la voiture rouge correspondait à la cinquième colonne.

Par la suite, on devait implémenter un constructeur de la classe *State* chargé de construire un nouvel état en fonction d'un état préalable et d'un mouvement d'une voiture. Ainsi, nous n'avions qu'à assigner à l'état précédent les diverses valeurs requises et à mettre à jour le nouvel état en déplaçant la voiture nécessaire.

Mouvements possibles

Cette section consistait en premier lieu en l'implémentation de la méthode *initFree*, chargée de déterminer les cases du stationnement non occupées par une voiture. Pour ce faire, nous avons tout d'abord initialisé l'ensemble de la matrice à *True* (libre). Par la suite, à l'intérieur d'une boucle dans laquelle on passait toutes les voitures comprises dans le stationnement, on déterminait si chaque voiture était positionné de façon horizontale ou verticale pour ensuite déterminer la rangée (ou colonne) sur laquelle elle se déplaçait ainsi que leur position relative dans cette rangée (ou colonne). Par la suite, on octroyait la valeur *False* (non libre) à la position de l'automobile ainsi qu'à toutes les cases suivantes correspondant à sa longueur.

Il nous était ensuite demandé de compléter la méthodes *moves* qui devait retourner l'ensemble des mouvements possible en fonction d'un état donné. Ainsi, cette fonction débutait en appelant *initFree* et bouclait ensuite sur l'ensemble des voitures afin de déterminer si la voiture pouvait bouger soit à gauche ou à droite (horizontale) ou soit en haut ou en bas (verticale). Le fonctionnement de cette méthode était très

semblable à celui de *initFree* à l'exception que l'on devait toujours vérifier que l'on ne faisait pas d'accès à l'extérieur du tableau. En effet, si par exemple une voiture horizontale était à l'extrémité gauche du stationnement et que l'on tentait de vérifier si elle pouvait se déplacer vers la gauche, on avait une erreur puisque l'on tentait d'accéder à une case qui n'était pas dans le tableau.

À la recherche d'une solution

C'est dans cette section que débutait réellement l'implémentation d'un algorithme pour résoudre le problème. Ainsi, il nous était demandé de compléter la fonction *solve* de la classe *State* en y implémentant un algorithme de fouille en largeur. On commençait donc par rajouter dans une liste l'état initial puis, tant et aussi longtemps que cette liste n'était pas vide, on y enlevait l'état en tête de file. Si cet état n'était pas final, on rajoutait à la fin de la liste l'ensemble des états que l'on pouvait obtenir à partir de l'état que l'on venait de retirer. De plus, on maintenait une liste (un *hash set*) des états déjà visités pour éviter des repasser plusieurs fois par un même état.

Récupérons une solution

Par la suite, on nous demandait de compléter la fonction *printSolution* qui était tout simplement chargée d'afficher le nombre de mouvements requis pour résoudre un problème ainsi que l'ensemble des mouvements effectués. Puisque cette fonction débutait avec l'état final, on devait utiliser *prev* qui représentait un état précédent. Ainsi, en passant tous les états précédents l'état final et en déterminant grâce à quatre *if* le mouvement effectué et la voiture en cause, on pouvait remplir une liste contenant l'ensemble des mouvements. Par la suite, il ne suffisait que d'imprimer cette liste en partant de la fin pour avoir les mouvements dans le bon ordre.

Autre approche

La dernière partie du travail consistait en l'implémentation d'un nouvel algorithme, toujours dans l'optique de résoudre le même problème. Ainsi, on nous demandait, à travers l'implémentation d'un algorithme *A**, d'ajouter des composantes d'intelligence artificielle dans la résolution du problème. Pour ce faire, nous devions tout d'abord implémenter le même algorithme que précédemment, soit un algorithme de recherche en largeur, mais en utilisant cette fois-ci une *priorityQueue* à la place d'une liste *FIFO*. Cette file de priorité permettait de comparer chacun des éléments deux à deux afin de déterminer, selon une fonction préétablie, lequel état devrait être visité en premier.

Par la suite, nous avons dû implémenter deux fonctions pour les utiliser avec la *priorityQueue* afin de déterminer quel état devrait être visité en priorité. Ainsi, la première fonction, *estimee1*, ne faisait que grossièrement estimer en combien de coups la voiture rouge pouvait sortir. Pour ce faire, on n'avait qu'à soustraire à la position de sortie (4) la position de la voiture rouge. Cependant, cette fonction était très simpliste puisqu'elle ne prenait absolument pas en compte le fait que des voitures pouvaient se trouver sur le chemin de la voiture rouge et de la sortie.

Ainsi, dans la deuxième fonction, *estimee2*, on remédiait à ce problème en additionnant au résultat de *estimee1* le nombre de voitures présentent entre la voiture rouge et la sortie. Pour ce faire, on n'avait qu'à parcourir toutes les automobiles, à déterminer celles qui étaient verticales, à s'assurer que ces voitures se retrouvaient dans une colonne situé en avant de la fin de la voiture rouge (la position de la voiture rouge additionné de sa longueur) et finalement que ces voitures avait un point d'intersection avec la rangée de sortie (2).

Question # 1 (comparaison des algorithmes avec *estimee1*)

Tableau comparatif entre l'algorithme en largeur et A avec estimee1*

	Solve 22	**Solve 1**	**Solve 40**
Largeur	39 ms / 4750 états	16 ms / 1068 états	26 ms / 2995 états
A* avec estimee1	48 ms / 4087 états	21 ms / 1035 états	38 ms / 2949 états

Tel qu'on peut le voir dans le tableau précédent, l'algorithme en largeur semble toujours prendre légèrement moins de temps à s'exécuter que l'algorithme A*, mais nécessite toujours un plus grand nombre d'états. Cela s'explique probablement par le fait que l'algorithme A* est plus efficace. Cependant, puisque l'algorithme A* doit appeler la fonction *estimee1* à chaque construction d'un nouvel état et puisqu'il fait usage d'une *priorityQueue* qui elle fait appel à un comparateur pour déterminer quel état devrait être traité en premier, il n'est pas surprenant que son exécution soit légèrement plus longue.

Question # 2 (comparaisons des algorithmes avec *estimee2*)

Tableau comparatif entre l'algorithme en largeur et A avec estimee2*

	Solve 22	**Solve 1**	**Solve 40**
Largeur	39 ms / 4750 états	16 ms / 1068 états	26 ms / 2995 états
A* avec estimee2	49 ms / 3681 états	24 ms / 990 états	40 ms / 2781 états

Exactement de la même façon que dans le cas de l'algorithme A* avec *estimee1*, l'algorithme A* avec *estimee2* prend plus de temps d'exécution mais moins d'états que l'algorithme de recherche en largeur. De plus, on peut voir que A* avec *estimee2* prend légèrement plus de temps que A* avec *estimee1* mais nécessite toujours également

moins d'états. Cela s'explique avec le fait que la fonction *estimee2* est plus complexe que *estimee1*, résultant en un plus grand temps d'exécution, mais également en un moins grand nombre d'états puisqu'elle est plus efficace pour déterminer quel état devrait être traité de façon prioritaire.

Conclusion

Tableau comparatif du ratio temps / nombre d'états entre l'algorithme en largeur, A avec estimee1 et A* avec estimee2*

	Solve 22	**Solve 1**	**Solve 40**
Largeur	0.008210526	0.014981273	0.008681135
A* avec estimee1	0.011744556	0.020289855	0.012885724
A* avec estimee2	0.0133116	0.024242424	0.014383315

En somme, et à l'aide du tableau récapitulatif ci-dessus (temps / nombre d'états), on voit clairement une amélioration de l'efficacité lorsque l'on passe de l'algorithme de recherche en largeur à un algorithme A* avec *estimee1*. De la même façon, on dénote une amélioration de performance lorsque l'on passe de l'algorithme A* avec *estimee1* à l'algorithme A* avec *estimee2.*